Building Adaptive E-Catalog Communities Based on User Interaction Patterns

Hye-young Paik and Boualem Benatallah, *University of New South Wales*

On the basis of observed customer interaction patterns, the WebCatalog^{Pers} system creates integrated product catalogs that continuously adapt and can be restructured in a dynamic environment.

In recent years, integration of *e-catalogs* has gained considerable momentum because of the emergence of online shopping portals, the increased demand for information exchange between trading partners, the prevalence of mergers and acquisitions, and so on. An e-catalog is any product catalog that you can search and query online, such as at www.dell.com or Amazon.com.

Most solutions to the problem of organizing and integrating e-catalogs use a category-based hierarchy to structure a product catalog in a "one view fits all" fashion. This hierarchy is often determined by a system designer, who usually has a priori expectations of how customers will explore catalogs. However, customers might have different expectations. To minimize the gap between these expectations, designers should consider how customers actually use the catalogs. For example, for a computer parts catalog, assume that many users always access the product category RAM right after the category CPU. If the administrator merges the two categories to form a new category CPU&RAM, users need to visit the new category only once for information on both products.

We propose a user-centric technique for restructuring integrated e-catalogs. We've implemented this technique in WebCatalog^{Pers},[1] a system that integrates online product catalogs. WebCatalog^{Pers} aims to continuously improve the organization of catalogs by responding to the ways customers navigate them. The resulting integrated catalogs continuously adapt and can be restructured in a dynamic environment.

WebCatalog^{Pers} design overview

Our technique uses a catalog integration framework that originated in the WebFindIt project.[2] The framework rests on the concept of *catalog communities*. A catalog community (just *community* throughout the rest of this article) is a container of catalogs that offer products of a common domain (for example, the Laptops community). It provides a description of desired products without referring to specific sellers. Figure 1 illustrates a community for the Computers and Related Services domain.

Relationships between communities fall into two types. *SubCommunityOf* relationships represent specialization between two communities' domains (for example, Printer is a subcommunity of Peripherals). We assume that each community has at most one supercommunity.

PeerCommunityOf relationships act as a referral mechanism. When a user cannot find (or is not satisfied with) information from a community, she or he can refer to other communities that the community considers as its peers (for example, Display is a peer community of VideoCard). We do not assume that the opposite systematically holds (for example, that VideoCard is a peer community of Display). A weight (a real value between 0 and 1) attached to each PeerCommunityOf relationship represents the degree of relevancy as a peer. Communities can also forward queries to each other via a PeerCommunityOf relationship.

We call such an organization of communities an *eCatalogs-Net*. Any community that is not a subcommunity of any other community is related to AllCatalog via a SubCommunityOf relationship. Each community has a set of attributes with which users can query the underlying catalogs. We refer to this set as *community product attributes* (community attributes, for short). For example, the CD Readers and Writers community would have community attributes such as Maker, ReadWriteSpeed, and Price.

For product sellers to be accessible through a com-

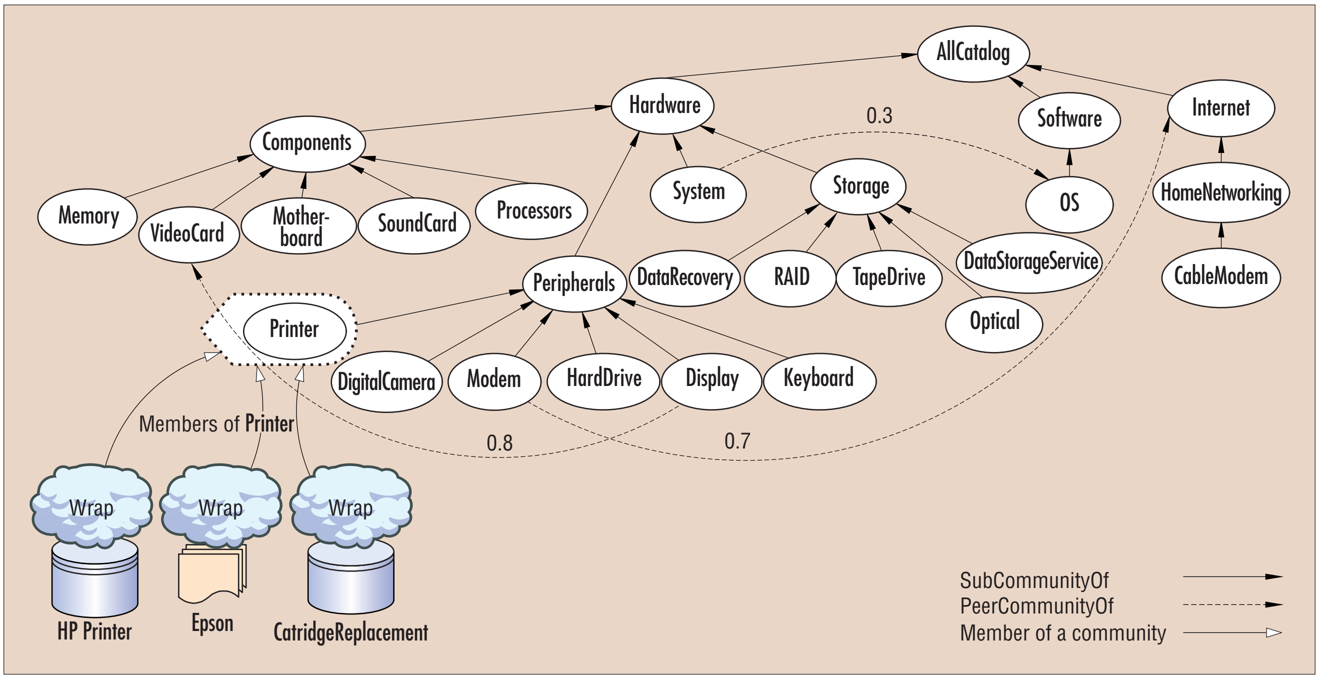


Figure 1. eCatalogs-Net: organizing a catalog community for the Computers and Related Services domain.

munity, they must register their catalogs with the community. This involves providing a *wrapper*, an *exported interface*, and a *source–community mapping*. The wrapper translates WebCatalog^{Pers} queries to local queries and translates the local queries' output back to the WebCatalog^{Pers} format. The exported interface defines the local product attributes for querying information at the local catalog. A local catalog supplier also should provide operations such as ordering or paying.

Users might use a community to express *global queries*, which require extracting and combining product attributes from multiple underlying product catalogs (for example, a price comparison). Global querying uses community attributes that do not directly correspond to product attributes. So, the source–community mapping defines the mapping between local product attributes and community attributes.

Adaptation

This process has three main components: a *catalog navigation and access model*, *predefined interaction sequences*, and *restructuring operations*.

Catalog navigation and access model

WebCatalog^{Pers} users will likely perform a two-step information-seeking activity:

1. Navigating communities to locate product catalogs (for example, finding communities that are relevant to selling laptops)
2. Querying selected communities or catalogs for product information (for example, comparing product prices)

Users will have a specific task to achieve when using product catalogs (for example, finding products they wish to purchase or investigating a category of products). We assume that they use the following strategy. They start at AllCatalog or at a specific community (if they know the community's location). If the current community is not the target community, they follow this procedure:

1. If any of the current community's SubCommunityOf relationships seem likely to lead to the target community, follow the relationship that appears most likely to lead to the target community.
2. Otherwise, if any of the current community's PeerCommunityOf relationships seem likely to lead to the target community, follow the relationship that appears most likely to lead to the target community.
3. Otherwise, either backtrack and follow one of the current community's SuperCommunityOf relationships or give up.

(Ramakrishnan Shrikant and Yinghui Yang use a similar strategy for browsing and searching Web documents—see the "Related Work" sidebar.)

On the basis of this user behavior, we define $\mathcal{A}$, the set of permissible catalog interaction actions for exploring an eCatalogs-Net (see Table 1). By modeling user interaction actions, the system can capture them for future use. (This article does not focus on specifying transactional operations—for example, ordering or payment for products. We assume that the local catalog supplier processes such operations via the exported interfaces. A detailed description of provisioning operations in the context of Web services appears elsewhere.[3])

Every time a user invokes a permissible action at a community, WebCatalog^{Pers} enters that event in the system log file. WebCatalog^{Pers} later transforms the log file into a suitable format for the system to use. This restructuring organizes the log into *sessions* and identifies all a community's product attributes that a query has selected. A session in WebCatalog^{Pers} is an ordered sequence of one user's actions, where the time difference between any two consecutive actions is within an administrator-defined time threshold, $T_{threshold}$.

Predefined interaction sequences

Predefined interaction sequences represent

Related work

Two major areas of related research are *building adaptive Web sites* and *navigation mining techniques for Web content personalization*.

Building adaptive Web sites

Mike Perkowitz and Oren Etzioni developed a methodology that automatically constructs index pages that supplement an existing organization by looking at co-occurring pages.[1] So, users can easily locate pages that are conceptually and strictly related to one topic. Takehiro Nakayama, Hiroki Kato, and Yohei Yamane have proposed a technique that discovers the gap between a Web site designer's expectations and a user's behavior.[2] Their technique compares interpage conceptual relevance with interpage access co-occurrence. Ramakrishnan Srikant and Yinghui Yang developed an algorithm to identify a Web page's expected locations and create a link from the expected location to the page.[3] Masashi Toyoda and Masaru Kitsuregawa's method extracts related pages from a given page using link analysis to create a Web community.[4] They also derive a Web community chart that connects related Web communities (that is, a map of Web communities) by clustering similarcommunities.

Although this area's basic principles are complementary to our work, most approaches deal only with Web pages, whereas our approach deals with *communities*. Communities are individual, autonomous entities (rather than a network of Web pages) with which users and community members can have various interactions (submitting queries, invoking operations, registering, and so on).

Navigation mining techniques

In the area of mining access patterns, Anindya Datta and his colleagues employ Web usage mining to dynamically predict a user's next behavior and make a recommendation.[5] José Borges and Mark Levene use Hypertext Probabilistic Grammar to predict the user's navigation path.[6] In a typical sequence of Web usage mining, an access pattern is a sequence of visited Web documents that occurs frequently. Such patterns extract frequently visited nodes, or nodes that are visited together, but they do not reflect how users navigate the imposed structure. Bettina Berendt and Myra Spiliopoulou propose their Web Usage Mining system to evaluate the effectiveness of Web site organization.[7] WUM uses the concept of *g-sequence* (generalized sequence) to model the sequence of user navigation. Sequences normally depict a single user's behavior as a vector of adjacent page requests. A g-sequence can depict navigational behavior patterns of many users using a wild card (*) and a range operator ([n;m]). We use a similar concept to model sequences of user interaction actions.

David Yang and his colleagues use decision trees to automatically construct catalogs on the basis of the product item popularity (that is, frequency of visits) and weighted product attributes.[8] Their construction algorithm minimizes the depth of the product hierarchy (which is a tree), pushing the popular product items or attributes to upper levels so that customers can find them easily (with fewer clicks). However, they do not discuss ongoing adaptivity for the catalogs. Also, WebFindIt considers addition and deletion of links between communities.[9] However, the mechanism is based on link-monitoring agents, which are different from mining access patterns.

References

1. M. Perkowitz and O. Etzioni, "Adaptive Web Sites," *Comm. ACM*, vol. 43, no. 8, Aug. 2000, pp. 152–158.
2. T. Nakayma, H. Kato, and Y. Yamane, "Discovering the Gap between Web Site Designers' Expectations and User's Behaviour, *Proc. 9th Int'l World Wide Web Conf.*, Foretec Seminars, Reston, Va., 2000, pp. 811–822; www9.org/w9cdrom/index.html.
3. R. Srikant and Y. Yang, "Mining Web Logs to Improve Website Organization," *Proc. 10th Int'l World Wide Web Conf.*, WWW10 Ltd., Hong Kong, 2001, pp. 430–437; www10.org/cdrom/papers/frame.html.
4. M. Toyoda and M. Kitsuregawa, "A Web Community Chart for Navigating Related Communities," poster presented at 10th Int'l World Wide Web Conf., WWW10 Ltd., Hong Kong, 2001, www10.org/cdrom/posters/frame.html.
5. A. Datta et al., "An Architecture to Support Scalable Online Personalization on the Web," *VLDB J.*, vol. 10, no. 1, 2001, pp. 104–117.
6. J. Borges and M. Levene, "Data Mining of User Navigation Patterns," *Proc. Workshop Web Usage Analysis and User Profiling* (WEBKDD 99), ACM Press, New York, 1999, pp. 92–111; www.acm.org/sigkdd/proceedings/webkdd99/toconline.htm.
7. B. Berendt and M. Spiliopoulou, "Analysis of Navigation Behaviour in Web Sites Integrating Multiple Information Systems," *VLDB J.*, vol. 9, no. 1, 2000, pp. 56–75.
8. D. Yang et al., "Construction of Online Catalog Topologies Using Decision Trees," *Proc. 2nd Int'l Workshop Advance Issues of E-Commerce and Web-Based Information Systems* (WECWIS 2000), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 223–230.
9. A. Bouguettaya et al., "Supporting Dynamic Interactions among Web-Based Information Sources," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 5, Sept./Oct. 2000, pp. 779–801.

foreseeable user interaction behavior. We use PISs to identify situations where restructuring might improve an eCatalogs-Net's organization. Any frequently occurring sequence of actions should be recognized as a recurring user interaction pattern. Such patterns can suggest a restructuring operation. We define a PIS as follows:

A PIS of length n (n > 0) is a vector of ordered user actions PIS $= \langle a_1, a_2, \ldots, a_n \rangle$, *where* $a_i \in \mathcal{A}$ (see Table 1) ($i = 1, \ldots, n$).

For a given PIS, a session s might exist that contains the exact order of actions in the PIS. WebCatalog^{Pers} matches a PIS against each session in the processed log file to check whether the sequence exists in the session.

The number of occurrences of a PIS in the file helps determine whether restructuring is necessary (we discuss this in more detail later).

We now look in detail at PISs for merging and splitting communities; Figure 2 lists other patterns. When defining a pattern, we use SubmitQuery as the most appropriate action for indicating a user's strong interests in a

Table 1. The set of permissible catalog interaction actions ($\mathcal{A}$) in eCatalogs-Net.

Action name	Description
NavigateToSub(Community c)	The user goes from the current community to one of its subcommunities **c**.
NavigateToSuper()	The user goes from the current community to its supercommunity.
NavigateToPeer(Community c)	The user goes from the current community to one of its peer communities **c**.
LeaveCatalogCommunity()	The user leaves the current community (she or he is taken to AllCatalog).
ShowMembers(Constraint s)	The user requests to see members of the current community satisfying the constraint **s**.
SubmitQuery(Query q)	The user submits a query **q** to the current community. **q** could be a global query, which uses the community attributes, or a source query, which concerns one community member.

community. However, an administrator might choose other actions (or define new ones—for example, PurchaseItem) for that purpose. (Even though the actions in Table 1 do not include source community parameters, we add them for clarity when defining PISs.)

Merging communities. $PIS_{GenericMerge}$ is a generic sequence that describes situations where merging of communities might be beneficial. We define it as follows:

PISGenericMerge, *which represents situations where two communities are always queried together, is*

$$PISGenericMerge = \langle SubmitQuery(c_i, q_1), a_1, \ldots, a_n, SubmitQuery(c_j, q_2)\rangle,$$

where $c_i, c_j \in N$, $a_k \in$ {NavigateToSub, NavigateToSuper, NavigateToPeer} *(*$k = 1, \ldots, n$*), and* q_1, q_2 *are global query attributes (that is, community attributes).*

$PIS_{GenericMerge}$ captures interaction sequences where users in a community c_i submit a query and then perform several navigation actions to reach a community c_j, from where they submit another query.

Figure 3 presents three particular cases of $PIS_{GenericMerge}$. Figure 3a shows a particular case where two subcommunities of the same supercommunity are always accessed together (without using any PeerCommunityOf relationship). In Figure 3b, a catalog community and its supercommunity are always queried together. Figure 3c shows a situation similar to that in Figure 3a: two communities are always accessed together, and they are linked via a PeerCommunityOf relationship.

Splitting a community. A community may be split if a subset of community attributes are always queried together and the subset by itself can represent a specific domain. PIS_{split} identifies a subset of attributes that are always queried together. In this pattern, an administrator has a specific community in mind (c_i) that she or he wants to examine for possible splitting and a set of attributes she or he predicts will be queried together.

We define PIS_{split} as follows:

Deleting a community: Identify a community that users constantly leave without performing any further action.

$$PIS_{delComm} = \langle a(c_i, c_j), LeaveCatalogCommunity(c_j)\rangle, \text{ where } c_i, c_j \in N, (c_i, c_j) \in E_1 \cup E_1^{-1}, \text{ and } a \in \{NavigateToSub, NavigateToSuper\}.$$

Merging communities (Case 1): Identify two subcommunities of the same supercommunity that are always accessed together (not via PeerCommunityOf).

$$PIS_{merge1} = \langle SubmitQuery(c_i,q_1), NavigateToSuper(c_i, c_k), NavigateToSub(c_k, c_j), SubmitQuery(c_j, q_2)\rangle, \text{ where } c_i, c_j, c_k \in N \text{ and } (c_i, c_k), (c_j, c_k) \in E_1.$$

Merging communities (Case 2): Identify a community and its supercommunity that are always queried together.

$$PIS_{merge2} = \langle SubmitQuery(c_i, q_1), a(c_i, c_j), SubmitQuery(c_j, q_2)\rangle, \text{ where } a \in \{NavigateToSuper, NavigateToSub\}, c_i, c_j \in N, \text{ and } (c_i, c_j) \in E_1 \cup E_1^{-1}.$$

Merging communities (Case 3): This PIS is the same as PIS_{merge} except it uses NavigateToPeer instead of NavigateToSub or NavigateToSuper.

$$PIS_{merge3} = \langle SubmitQuery(c_i, q_1), NavigateToPeer(c_i, c_j), SubmitQuery(c_j, q_2) \rangle, \text{ where } c_i, c_j \in N \text{ and } (c_i, c_j) \in E_2.$$

Upgrading the weight of a PeerCommunityOf: This PIS consolidates the relationship's relevancy. Assume that many users navigate from community c_i, via a PeerCommunityOf relationship, to community c_j and submit a query to c_j. This indicates that the PeerCommunityOf relationship from c_i to c_j positively contributed to finding the target community.

$$PIS_{upgrade} = \langle NavigateToPeer(c_i, c_j), SubmitQuery(c_j, q)\rangle, \text{ where } c_i, c_j \in N, (c_i, c_j) \in E_2, \text{ and } q \text{ is global query attributes.}$$

Downgrading the weight of a PeerCommunityOf: Assume that many users who followed a PeerCommunityOf relationship and arrived at a community c_j ultimately leave the community without performing any further action. This might indicate that c_j is not relevant to these users. To leave c_j, use LeaveCatalogCommunity or NavigateToPeer.

$$PIS_{downByLeave} = \langle NavigateToPeer(c_i, c_j), LeaveCatalogCommunity(c_j)\rangle, \text{ where } c_i, c_j \in N, \text{ and } (c_i, c_j) \in E_2.$$
$$PIS_{downByPeer} = \langle NavigateToPeer(c_i, c_j), NavigateToPeer(c_j, c_i)\rangle, \text{ where } c_i, c_j \in N, \text{ and } (c_i, c_j), (c_j, c_i) \in E_2.$$

Figure 2. Some predefined interaction sequences.

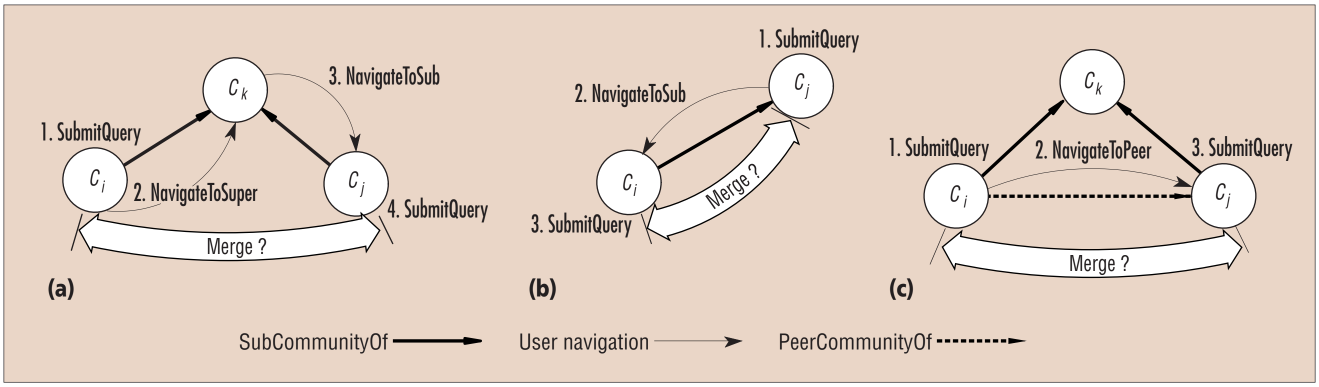


Figure 3. Three cases of $PIS_{GenericMerge}$: (a) two subcommunities of the same supercommunity are always accessed together (without using any PeerCommunityOf relationship); (b) a catalog community and its supercommunity are always queried together; (c) two communities are always accessed together, and they are linked via a PeerCommunityOf relationship.

PIS_{split}, which represents the pattern for splitting a community, is

PIS_{split} = ⟨SubmitQuery(c_i, "$attr_1$, …, $attr_n$")⟩,

where $c_i \in N$ and $attr_1, \ldots, attr_n$ are community attributes in c_i that will likely be queried together.

Restructuring operations

These operations can, for example, change the relationships between communities, remove a community, or merge communities. An administrator can perform them at her or his discretion. However, PISs provide means to observe the user's interaction patterns. The observation will help decide which operation will improve the organization of communities.

Figure 4 shows the primitive and high-level restructuring operations. Each high-level operation constitutes a sequence of primitive operations. For space reasons, we describe only the operations for moving, merging, and splitting communities.

Moving a community. The operation **moveCatComm()** relocates a community c from one place to another by changing its supercommunity. An administrator uses this operation, for example, when she or he is convinced that the current supercommunity of c does not represent properly the domain of products in c.

For example, in Figure 1, assume that the HardDrive community is a subcommunity of Peripherals but that user navigation behavior shows that the Storage community is a more suitable supercommunity for HardDrive. This might suggest that moving HardDrive to Storage will be beneficial. When a community c moves, its subcommunities move with it. This assumption creates less overhead because the change does not affect those subcommunities.

Merging communities. The operation **mergeCatComm()** merges two communities, c and c′, that have the same supercommunity. (In this article, we consider merging only two communities, but you can generalize the operation to more communities.) An administrator uses this operation, for example, when she or he observes that users almost always access c and c′ together. So, merging these two communities will be beneficial because most users will not have to visit two separate communities each time. One example of this situation is the merging of CPU and RAM that

setCatalogName(Communityc, Stringn): Set the name of c to n.
addPeer(Communityc$_i$, Communityc$_j$): Add PeerCommunityOf from c_i to c_j.
delPeer(Communityc$_i$, Communityc$_j$): Delete PeerCommunityOf from c_i to c_j.
updatePeer(Communityc$_i$, Communityc$_j$, Weightw): Update the weight of PeerCommunityOf from c_i to c_j by w.
addSub(Communityc$_i$, Communityc$_j$): Add SubCommunityOf from c_i to c_j.
delSub(Communityc$_i$, Communityc$_j$): Delete SubCommunityOf from c_i to c_j.
createCatComm(Namen, GeneralInfogi, Membersm, CommunityProductAttrgs): Create a new community with the information given. The **addSub** operation must follow **createCatComm**.
(a)

mergeCatComm(Communityc$_i$, Communityc$_j$, Namen): Merge two existing communities c_i and c_j, and set the name of the new community to n.
splitCatComm(Communityc, GeneralInfogic, Namen, GeneralInfogi, CommunityProductAttrcpa, Queryq, setOfCommunitiessub): Split community c into two separate communities. **gic** contains a new specification of **GeneralInfo** for c. **n**, **gi**, and **cpa** contain specifications of the new community (**Name**, **GeneralInfo**, and **CommunityProductAttr**, respectively). **q** is a query that the operation will use to select members to move from c to the new community. **sub** is a set of subcommunities to move to the new community.
delCatComm(Communityc): Remove the community c from eCatalogs-Net. This operation is used, for example, when a community becomes obsolete (for example, it has no useful existence inside an eCatalogs-Net).
moveCatComm(Communityc$_i$,Communityc$_j$): Move c_i to new supercommunity c_j.
(b)

Figure 4. eCatalogs-Net restructuring operations: (a) primitive; (b) high-level.

we mentioned in the introduction.

Splitting a community. The operation splitCatComm() splits a community into two separate communities. An administrator uses this operation, for example, when she or he observes that the community represents a domain (described by community attributes) that is divisible into subdomains.

For example, assume that when users submit queries to the MobileComputing community, which includes products such as laptops, handheld computers, portable disks, and cellular phones, we see a clear division between attributes relating to handheld computers (such as PDAMemory or PDAOperatingSystem) and attributes relating to laptop computers (such as Modem, Ethernet, BatteryLife, or Graphics). In other words, all attributes relating to handheld computers are always queried together, and all attributes relating to laptops are always queried together. In this case, we decide that we can divide the community into two subdomains, one representing handheld computers and one representing laptops.

Pattern frequency and relevance

Whether restructuring is suitable for a PIS depends on its *frequency* and *relevance*.

Frequency. We use frequency to decide whether the result of a match between a user navigation sequence and a PIS is significant enough to consider restructuring. We define the frequency of a PIS as

Freq(PIS) = *number of occurrences of a PIS in the processed log file.*

Relevance. Two special situations arise from using PISs. First, *conflicting patterns* occur when one pattern suggests a certain restructuring operation but another pattern suggests a different operation on the same relationships or communities. For instance, $\text{PIS}_{\text{upgrade}}$ might suggest upgrading the weight of the PeerCommunityOf relationship between communities A and B. However, $\text{PIS}_{\text{downbyPeer}}$ might suggest downgrading the relationship.

The second situation is that of *consolidating patterns*, which reinforce each other's findings. For instance, suppose that $\text{PIS}_{\text{downbyLeave}}$ suggests downgrading the PeerCommunityOf relationship between communities A and B. Knowing that $\text{PIS}_{\text{downbyPeer}}$ also suggests downgrading that relationship will provide assurance when administrators choose a restructuring operation.

We initially define a PIS's relevance as

$$\text{Relevance(PIS)} = \frac{\text{RF(cons)}}{\text{RF(conf)}} \cdot \log \frac{\text{RF(conf)}}{\text{RF(cons)}} \quad . \qquad (1)$$

RF(cons) is the relevance factor of all consolidating patterns:

$$\text{RF(cons)} = \frac{\text{Freq(PIS)} + \text{A}}{\text{Freq(PIS)}} \ , \qquad (2)$$

where A is the sum of the frequency of all consolidating patterns. RF(conf) is the relevance factor of all conflicting patterns:

$$\text{RF(conf)} = \frac{\text{Freq(PIS)} + \text{B}}{\text{Freq(PIS)}} \ , \qquad (3)$$

> We use frequency to decide whether the result of a match between a user navigation sequence and a PIS is significant enough to consider restructuring.

where B is the sum of the frequency of all conflicting patterns. Combining Equations 1 through 3 gives us

$$\text{Relevance(PIS)} = \frac{\text{Freq(PIS)} + \text{A}}{\text{Freq(PIS)} + \text{B}} \cdot \log \frac{\text{Freq(PIS)} + \text{A}}{\text{Freq(PIS)} + \text{B}} \quad . \qquad (4)$$

In Equation 4, (Freq(PIS) + A)/(Freq(PIS) + B) indicates the consolidating patterns' deviation from the conflicting patterns. Three relationships are possible:

- If (Freq(PIS) + A)/(Freq(PIS) + B) is close to 1—that is, $\text{A} \cong \text{B}$—the frequency of all consolidating patterns is almost equal to the frequency of all conflicting patterns.
- If (Freq(PIS) + A)/(Freq(PIS) + B) is less than 1—that is, $\text{A} < \text{B}$—then conflicting patterns occur more than consolidating ones.
- If (Freq(PIS) + A)/(Freq(PIS) + B) is greater than 1—that is $\text{A} > \text{B}$—then consolidating patterns occur more frequently than conflicting patterns.

For finding relevant patterns, we are interested in only the first and third relationships. So, we take the logarithm function because log((Freq(PIS) + A)/(Freq(PIS) + B)) is less than 0 for the second relationship. We add the weight Freq(PIS) + A to the measure to take into account how frequently the consolidating patterns occur. Another weight, 1/(Freq(PIS) + B)), indicates how valid the correlation is between consolidating and conflicting patterns. Thus, the higher Relevance(PIS) is, the more relevant the pattern is.

On the basis of the above definition of the relevance factors of consolidating and conflicting patterns, we obtain this definition of pattern relevance:

A pattern PIS is relevant if Relevance(PIS) $\geq \delta$, *where* δ (≥ 0) *is a threshold given by a system administrator.*

Implementation and evaluation

Figure 5 shows the main modules of WebCatalog$^{\text{Pers}}$. CatalogBuilder lets community creators build a community and gives product catalog providers methods to register their local product catalog. The wrapper classes in CatalogBuilder define methods that can translate global queries to local catalog queries and vice versa. The Web-based user interface provides navigation methods for customers to interact with eCatalogs-Net and lets catalog providers register their catalogs.

The classes in NavigationManager provide methods to manipulate an eCatalogs-Net (for example, add a new subcommunity or identify a community's peer communities) and to manipulate a single community (for example, name a community or identify a community's members). LogManager traps every catalog interaction action and its parameters for logging. WebCatalog$^{\text{Pers}}$ feeds the logged data to DataTransformer, which transforms data into a suitable format for the system, as we mentioned earlier. An administrator can operate this module periodically (for example, every week).

An administrator uses WebCatalogAdmin to restructure an eCatalogs-Net. The module interacts with PatternQueryEngine and eCatalogNetReorganiser. PatternQueryEngine processes queries for the PISs. We have implemented a set of ad hoc PL/SQL (Procedural Language/Structured Query Language) programs for querying the patterns we introduce in this article. They search the log file session by session for any sequence that matches the PIS and return the matching sequences with a frequency. eCatalogNetReorganiser mainly communicates with the

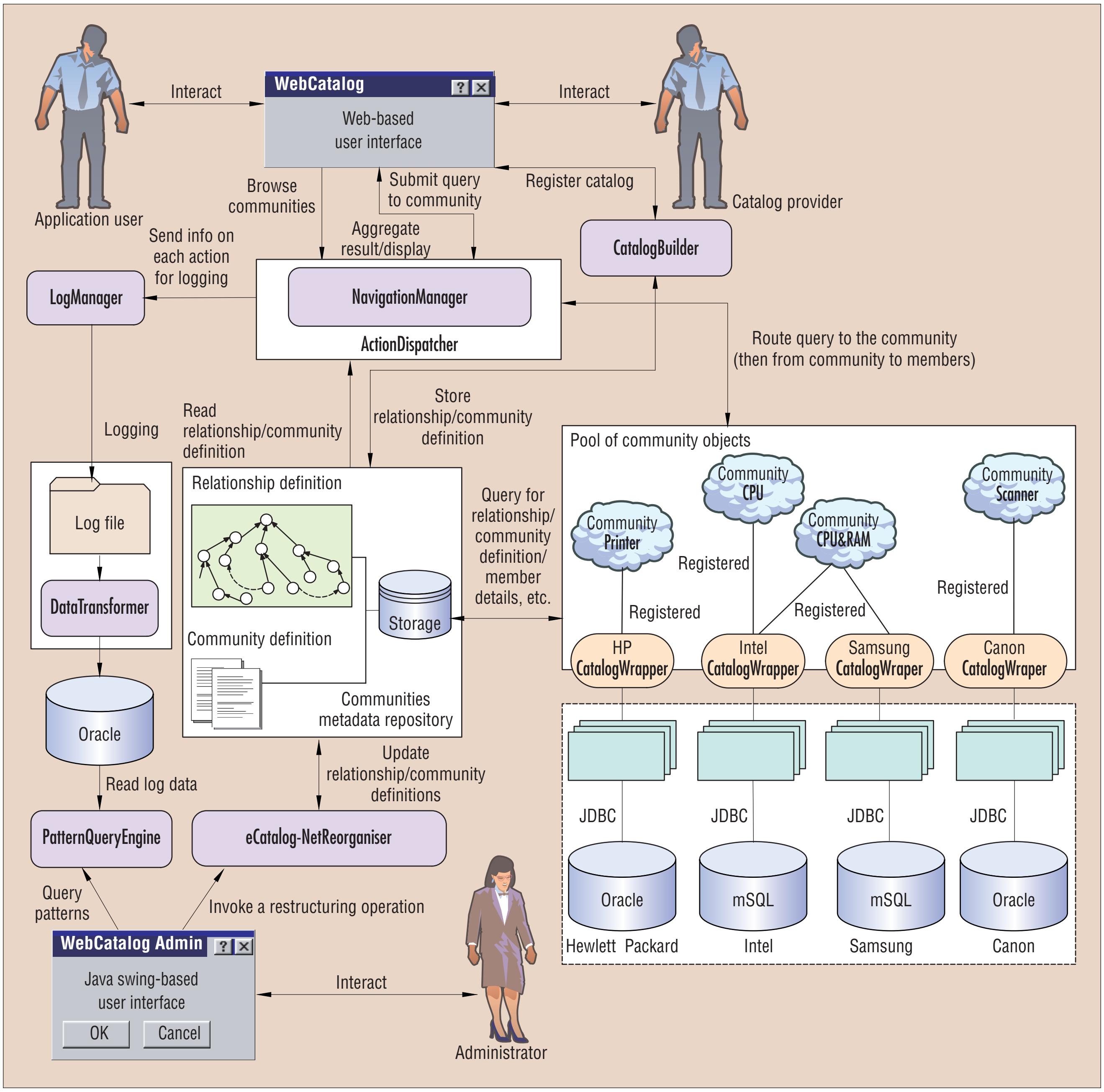


Figure 5. The WebCatalog^{Pers} architecture.

repository. An administrator can invoke any defined restructuring operation through this interface, which involves manipulating the relationship and community definitions.

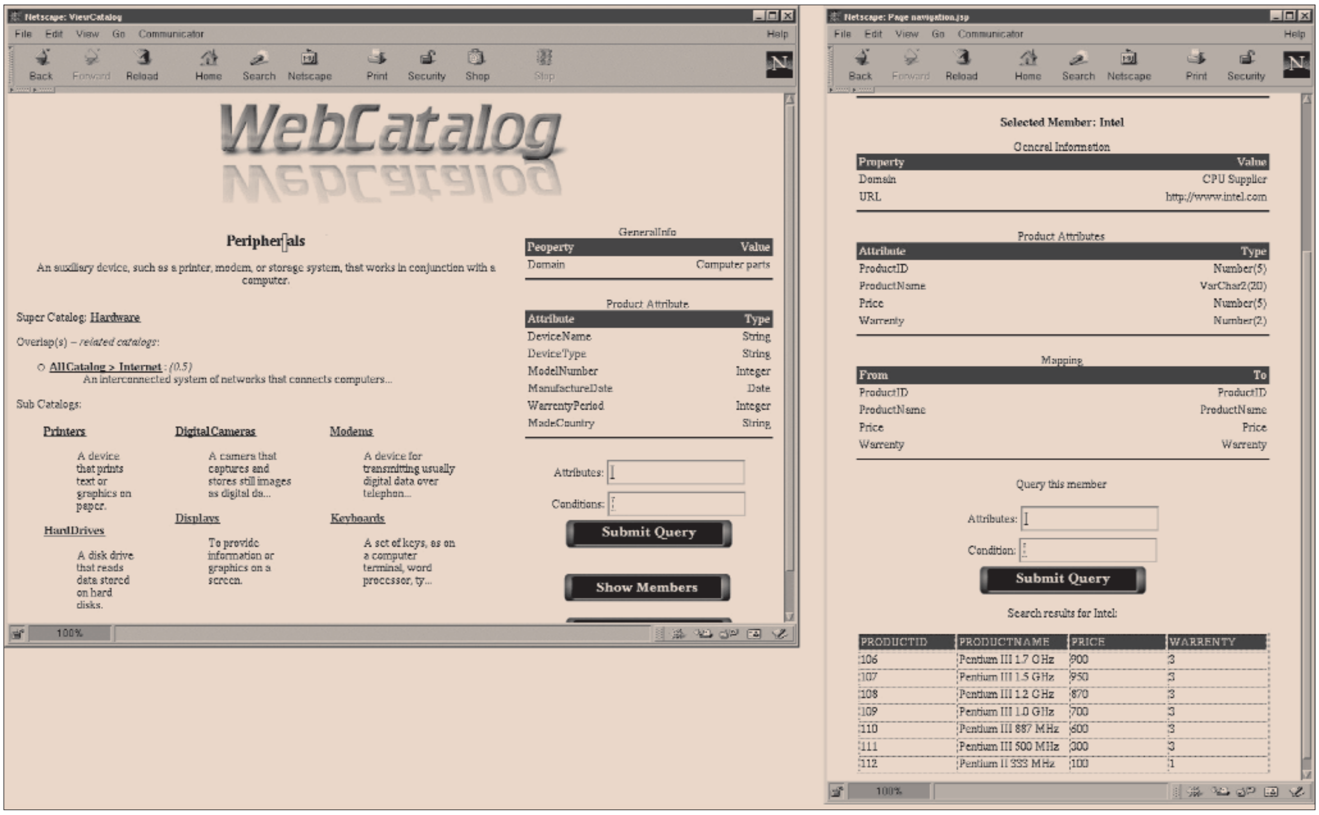
A product catalog's wrapper class (initiated when a product catalog is registered) is associated with a Java application that accesses the product catalog's local database and handles the translation between global and local queries. The application supports creating a community, registering a product catalog, and querying a product catalog through a community.

Figure 6a illustrates the main user interface, where users can start navigating through the communities. The interface presents relationships between communities as hyperlinks so that users can easily move between communities by clicking links. It displays general information and product attributes for the community the user is looking at. Using the product attributes, users can submit a global query, which will go to all the community's members. Figure 6b shows a screenshot of the system after a user has selected a community member and submitted a query to it. This screen also displays the query's result.

Evaluation framework

To demonstrate this architecture's viability, we took examples of product catalogs from the Computer and Related Services domain and created 27 communities (as in Figure 1), each with three to four members (that is, catalog providers).

In the evaluation, we measured improvement due to restructuring by comparing the


Netscape: ViewCatalog
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
WebCatalog
Submit Query
Show Members
100%
Netscape: Page navigation.jsp
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Submit Query
100%

Peripherals

An auxiliary device, such as a printer, modem, or storage system, that works in conjunction with a computer.

Super Catalog: Hardware

Overlap(s) – *related catalogs*:

- AllCatalog > Internet : *(0.5)*
  An interconnected system of networks that connects computers...

Sub Catalogs:

Printers
A device that prints text or graphics on paper.

HardDrives
A disk drive that reads data stored on hard disks.

DigitalCameras
A camera that captures and stores still images as digital da...

Displays
To provide information or graphics on a screen.

Modems
A device for transmitting usually digital data over telephon...

Keyboards
A set of keys, as on a computer terminal, word processor, ty...

GeneralInfo

Peoperty	Value
Domain	Computer parts

Product Attribute

Attribute	Type
DeviceName	String
DeviceType	String
ModelNumber	Integer
ManufactureDate	Date
WarrentyPeriod	Integer
MadeCountry	String

Attributes:

Conditions:

Selected Member: Intel

General Information

Property	Value
Domain	CPU Supplier
URL	http://www.intel.com

Product Attributes

Attribute	Type
ProductID	Number(5)
ProductName	VarChar2(20)
Price	Number(5)
Warrenty	Number(2)

Mapping

From	To
ProductID	ProductID
ProductName	ProductName
Price	Price
Warrenty	Warrenty

Query this member

Attributes:

Condition:

Search results for Intel:

PRODUCTID	PRODUCTNAME	PRICE	WARRENTY
106	Pentium III 1.7 GHz	900	3
107	Pentium III 1.5 GHz	950	3
108	Pentium III 1.2 GHz	870	3
109	Pentium III 1.0 GHz	700	3
110	Pentium III 887 MHz	600	3
111	Pentium III 500 MHz	300	3
112	Pentium II 333 MHz	100	1

Figure 6. The user interface for (a) navigation and (b) querying a member catalog.

number of (simulated) users who found what they were looking for before restructuring and after restructuring. This study's primary goal is to demonstrate that given the same constraint (in our initial studies, a limited number of mouse clicks), more users can find targets after restructuring.

We created *task agents* that played the role of customers who wanted to find information about the products. To create agents, we used the Java class AgentFactory. This class implements a software component comprising a container and a pool of objects that represent agents. The container is a process that, once created, runs continuously. It listens to a socket, through which it receives an instantiation message from a predefined script (that creates an agent). An agent interacts with a *community manager* module, which provides various methods for exploring the community relationships (for example, getSubCommunityOf() or getPeerCommunityOf()). The agent's search and query behavior uses the strategy we present in the section "Catalog navigation and access model."

Two kinds of information help the agents interact autonomously with communities. The first is the relationships between communities (that is, SubCommunityOf and PeerCommunityOf).

The second is a *likelihood table*. Every community in an eCatalogs-Net has an assigned *likelihood*, a number value that represents its degree of "closeness" (that is, relevance) to a target community. So, the higher the value, the more likely the community will lead the agent to the target. The likelihood table constitutes the list of likelihoods.

Having the likelihoods fixed in the table makes the agents' interaction sequence predictable. However, agents should be able to make spontaneous, irregular decisions, resulting in unpredictable behavior. So, we introduced a *variant factor* that will diverge a likelihood value. When an agent receives the likelihoods, it dynamically recalculates them according to the factor before starting navigation.

To run, the agent takes these inputs:

- The name of the file that contains the likelihood table
- The target community's name
- The maximum number of moves an agent can make before giving up

We limited the maximum number of moves to 14 for all experiments. The parameter VF represents the variant factor's value. We asked four people who are familiar with the domain to produce the likelihood tables. The likelihood values we used in the experiments took the average values of the four. We used three settings of VF: 15 percent, 10 percent, and 5 percent. The higher VF is, the bigger the deviation is from given likelihood values. (We also defined other experimental parameters related to the likelihood table, such as the number of tables participating in the experiment and the range of likelihood values. However, we present only the experiments with VF.)

Evaluation results

We investigated the effect of two restructuring operations (addPeer and moveCatComm) and VF in two scenarios.

The first involved a PeerCommunityOf relationship. For initial runs, we created 3,000 agents and tasked them with finding the community CableModem (see "Before" in Figure 7a). In the initial runs, approximately 28 percent of the agents who found the target followed the PeerCommunityOf relationship from Modem to Internet, using Internet and HomeNetworking as

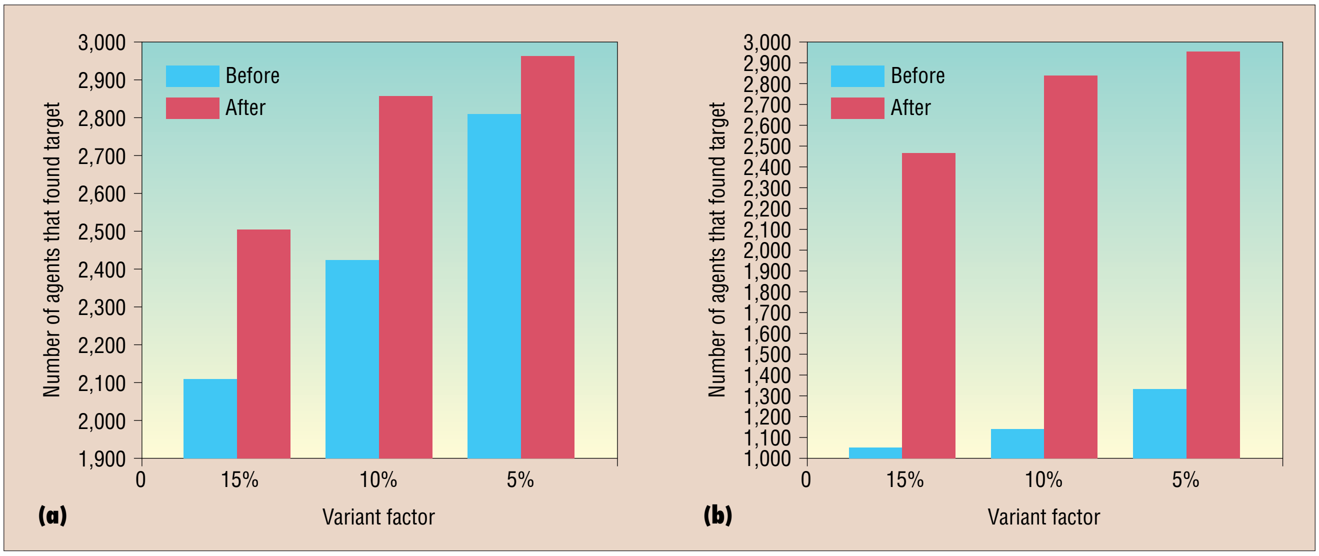


Figure 7. Varying the variant factor (VF) for two restructuring operations: (a) creating a peer community; (b) moving a community.

stopovers. We performed addPeer to create a new PeerCommunityOf relationship from Modem to CableModem. Then we ran the 3,000 agents again (see "After" in Figure 7a).

Figure 7a also shows that as VF decreases, more agents can find the targets, both before and after restructuring. This demonstrates that agents will probably find a target if their likelihood values are closer to the given likelihoods. Because the likelihoods we used described the relationships of communities relatively precisely, we interpret this result to mean that users whose understanding does not deviate much from that of domain experts will more likely find targets easily.

The biggest improvement after restructuring occurred when VF was 15 percent (that is, the highest deviation from given likelihoods). This indicates that restructuring an eCatalogs-Net provides the most benefits when the user's understanding deviates considerably from the expert's.

In the second scenario, we performed moveCatComm to move a community to a new supercommunity. Figure 7b shows the results.

Overall, restructuring produced obvious improvements, demonstrating that adaptive structuring of e-catalogs can give users easier, more streamlined navigation and search.

We are looking into the automatic inference of relationships between communities based on the comparison of their descriptions as well as description of their members. We also plan to develop data mining-based techniques for analyzing queries submitted to communities to find new relationships between communities as well as between communities and product catalogs.

Another research direction is the integration of WebCatalog^{Pers} with Web services. The objective is to use the restructuring techniques proposed in WebCatalog^{Pers} to effectively manage interactions between components of a composite service.[3] ■

Acknowledgments

The Australian Research Council's Discovery Grant DP0211207 partially supports Boualem Benatallah's research. We are especially grateful to Rachid Hamadi from the University of New South Wales Database Group for his valuable comments and suggestions.

References

1. H. Paik, B. Benetallah, and R. Hamadi, "User-Centric Adaptation of Dynamic E-Catalogs," *Proc. 4th Int'l Conf. Advanced Information Systems Eng.*, Springer-Verlag, Berlin, 2002, pp. 344–360.
2. A. Bouguettaya et al., "Supporting Dynamic Interactions among Web-based Information Sources," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 5, Sept./Oct. 2000, pp. 779–801.
3. B. Benatallah et al., "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services," *Proc. 18th Int'l Conf. Data Eng.*, IEEE CS Press, Los Alamitos, Calif., 2002, pp. 297–308.

The Authors

Hye-young Paik is a PhD candidate in computer science at the University of New South Wales. Her research interests include Web services, Web-accessible databases, and personalization issues in online communities. She received her bachelor's in natural science from Sungshin Women's University and her master's in information science from the University of New South Wales. She is a student member of the IEEE Computer Society and ACM. Contact her at the School of Computer Science and Eng., Univ. of New South Wales, Sydney 2052, NSW, Australia; hpaik@cse.unsw.edu.au.

Boualem Benatallah is a senior lecturer at the University of New South Wales. His latest work focuses on Web services, Web databases, and workflows. He received his MSc and PhD in computer science from IMAG (Grenoble Univ.). He is a member of the IEEE and ACM. Contact him at the School of Computer Science and Eng., Univ. of New South Wales, Sydney 2052, NSW, Australia; boualem@cse.unsw.edu.au; www.cse.unsw.edu.au/~boualem.